

DEBUGGER IMPACT REDUCTION THROUGH BREAKPOINT MOTION

Background of the Invention

1. Field of the Invention

The invention relates to debugger impact reduction through breakpoint motion.

2. Background Information

A debugger is a software tool used by programmers in software development to identify and correct program errors within a program. To support the debugger, information describing symbols and types in the program as well as information to map between source lines and the binary code may be provided by a compiler and/or an interpreter ("compiler"). This extra information, generally referred to as debugging information, enables the programmer to examine the types, variables and data structures by name and to follow the execution of the program through the source code.

With a debugger, programmers may "step" through program code statements one at a time, while the corresponding machine instructions are being executed. As programmers step through code, they may, inter alia, concurrently monitor changes in certain variables, which assists in identifying and correcting program errors.

With a debugger, a programmer may also set a breakpoint at a specific program position. When the program is executed, the debugger will stop execution of the program at any breakpoint encountered, and may display various programming elements, such as software variables for example, to assist in the debugging process.

Debuggers may also allow programmers to define and set conditional breakpoints. A conditional breakpoint is a breakpoint that has an associated Boolean expression. Upon encountering a conditional breakpoint, a debugger will only stop execution of the program if the associated Boolean expression is evaluated by the debugger to be TRUE (i.e., satisfied). For example, a conditional breakpoint may be BREAK 6 IF K < 999. Accordingly, a debugger would only stop execution of the program at the location of such a conditional breakpoint (here, line 6) if the variable K is less than the value 999 when the conditional breakpoint is encountered.

In today's software development environment, many new software applications are threaded. As such, they are designed to efficiently execute in multitasking or multiprocessing environments, which allow multiple threads, or streams of execution, to execute concurrently. With the increased frequency of threaded applications, programming timing errors are becoming more common than ever.

A significant problem with debugging program timing errors is that the introduction of breakpoints may dramatically affect program timing, which may make difficult or impossible the reproduction of program errors in the debugging environment. While conditional breakpoints may affect overall program timing less than normal breakpoints and program code stepping, they still require a short amount of time for their associated Boolean expressions to be evaluated. Although this short amount of time may be tolerable in some cases, where the conditional breakpoint is encountered a large number of times, program timing may be so significantly affected that reproducing program errors in a debugging environment may be difficult or impossible.

A frequently occurring situation in which a conditional breakpoint may be encountered a large number of times is where the breakpoint is set within a program loop. A program loop is a repetition within a program, and may iterate a large number of times. Therefore, where a conditional breakpoint is set within a program loop, program timing is more likely to be substantially affected, and accordingly, the difficulty or impossibility of reproducing program errors in the debugging environment becomes more likely.

Therefore, there is a need for debugger impact reduction.

Summary of the Invention

It is, therefore, a principal object of this invention to provide debugger impact reduction through breakpoint motion.

It is another object of the invention to provide debugger impact reduction through breakpoint motion that assists in minimizing the above mentioned problems.

These and other objects of the present invention are accomplished by debugger impact reduction through breakpoint motion as disclosed herein.

In an exemplary aspect of the invention, a method of reducing debugger impact through conditional breakpoint motion comprises: extracting, from an initial conditional breakpoint within a program loop, a first Boolean expression that is at least partially invariant within the loop; setting, at a pre-ICB (pre-initial conditional breakpoint, see infra) program position, a special conditional breakpoint that includes the first Boolean expression; removing the initial conditional breakpoint; and if the special conditional breakpoint is satisfied, reestablishing the initial conditional breakpoint.

In a further exemplary aspect of the invention, a method of reducing debugger impact through conditional breakpoint motion may further comprise: extracting, from the program code within the loop, a second Boolean expression that is invariant within the loop, where the special conditional breakpoint disjunctively includes the complement of the second Boolean expression, and the first Boolean expression is invariant within the loop when the second Boolean expression is satisfied.

In another exemplary aspect of the invention, a method of reducing debugger impact through conditional breakpoint motion may further comprise: setting, at one or more loop exit program positions, a reset breakpoint; and if one of the reset breakpoints is satisfied, removing the reset breakpoints and/or the initial conditional breakpoint.

In an alternative exemplary aspect of the invention, a method of reducing debugger impact through conditional breakpoint motion may comprise: extracting, from an initial conditional breakpoint within a program loop, a first Boolean expression that is at least partially invariant within the loop; setting, at a pre-ICB program position, a special conditional breakpoint that includes the complement of the first Boolean expression; and if the special conditional breakpoint is satisfied, removing the initial conditional breakpoint.

In a further alternative exemplary aspect of the invention, a method of reducing debugger impact through conditional breakpoint motion may further comprise: extracting, from the program code within the loop, a second Boolean expression that is invariant within the loop, where the special conditional breakpoint conjunctively includes the second Boolean expression, and the first Boolean expression is invariant within the loop when the second Boolean expression is satisfied.

In another alternative exemplary aspect of the invention, a method of reducing debugger impact through conditional breakpoint motion may further comprise: setting, at one or more loop exit program positions, a reset breakpoint; and if one of the reset breakpoints is satisfied, removing the reset breakpoints and/or reestablishing the initial conditional breakpoint.

Thus, the present invention, inter alia, creates a state that assists in avoiding unnecessary evaluations of a conditional breakpoint within a program loop.

Each of the above aspects may be embodied in a method, a debugger, and an article of manufacture comprising a computer readable program storage medium tangibly embodying one or more programs of instructions executable by a computer.

Brief Description of the Drawings

Figure 1 is a flow diagram illustrating debugger impact reduction through conditional breakpoint motion with optional reset breakpoints, according to a first class of embodiments of the invention.

Figure 2 is a flow diagram illustrating debugger impact reduction, with optional reset breakpoints, and having an additional Boolean expression extraction and a modified special conditional breakpoint setting, according to the first class of embodiments.

Figure 3 is a flow diagram illustrating debugger impact reduction with optional reset breakpoints, according to a second class of embodiments of the invention.

Figure 4 is a flow diagram illustrating debugger impact reduction, with optional reset breakpoints, and having an additional Boolean expression extraction

and a modified special conditional breakpoint setting, according to the second class of embodiments.

Figure 5 illustrates an exemplary set of program code used to describe the methods of Figures 1-4 in operation.

5 Figure 6 illustrates another exemplary set of program code used to describe the methods of Figures 1-4 in operation.

Detailed Description

10 The invention will now be described in more detail by way of example with reference to the illustrative embodiments shown in the accompanying figures. It should be kept in mind that the following described embodiments are only presented by way of example and should not be construed as limiting the inventive concept to any particular configuration or order.

15 As shown in Figure 1, in an exemplary aspect of the invention, a debugger (not shown) may effectuate conditional breakpoint motion through a first Boolean expression extraction (block 2a); a special conditional breakpoint setting (block 4a); an initial conditional breakpoint removal (block 6a); an optional reset breakpoint setting (block 8a); an initial conditional breakpoint reestablishment (block 10a); an optional subsequent initial conditional breakpoint removal (block 12a), and an
20 optional reset breakpoint removal (block 14a).

Figure 2 illustrates another exemplary aspect of the invention, in which a debugger (not shown) may effectuate conditional breakpoint motion through, inter alia, a second, additional, Boolean expression extraction (block 2b') and a modified special conditional breakpoint setting (block 4b).

25

As shown in Figure 3, in an alternative exemplary aspect of the invention, a debugger (not shown) may effectuate conditional breakpoint motion through a first Boolean expression extraction (block 2c); a special conditional breakpoint setting (block 4c); an optional reset breakpoint setting (block 8c); an initial conditional breakpoint removal (block 6c); an optional conditional breakpoint reestablishment (block 10c); and an optional reset breakpoint removal (block 14c).

Figure 4 illustrates another alternative exemplary aspect of the invention, in which a debugger (not shown) may effectuate conditional breakpoint motion through, inter alia, a second, additional, Boolean expression extraction (block 2d') and a modified special conditional breakpoint setting (block 4d).

Reference is now also made to Figures 5 and 6 to illustrate exemplary breakpoint motions by a debugger (not shown) according to the invention as illustrated in Figures 1-4. Figures 5 and 6 illustrate exemplary sets of program code containing exemplary conditional breakpoints within exemplary program loops.

In a first Boolean expression extraction (blocks 2a-d), a debugger extracts, from an initial conditional breakpoint 20 within a program loop, a first Boolean expression 22 (labeled as "BE_1" in Figures 5 and 6) that is at least partially invariant within the loop.

In Figure 5, initial conditional breakpoint 20 has been set at line 6, which is within a for-loop, and contains first Boolean expression 22 ("K > 999"), which is, in this example, completely invariant within the loop, as its evaluation will not be affected by code within the loop.

Figure 6 illustrates, inter alia, that first Boolean expression 22 may be a compound Boolean expression, which contains a plurality of propositions, but

nonetheless, may be treated as a single Boolean expression for evaluation purposes.

As shown in Figure 6, initial conditional breakpoint 20 has been set at line 7, which is within a while-loop, and contains first Boolean expression 22 ("(K > 999) && (K < 1050)"), which is partially invariant with the loop, as its evaluation is
 5 subject to change by code within the loop only upon satisfaction of a condition that is invariant within the loop (i.e., second Boolean expression 24, as discussed below).

Referring again to Figure 6, in a modified extraction step (blocks 2b' and 2d' of Figures 2 and 4), where first Boolean expression 22 is partially invariant within the loop, the debugger extracts from program code within the loop second Boolean
 10 expression 24 (labeled as "BE_2" in Figure 6) that is invariant within the loop, as its evaluation is not subject to change by code within the loop. In Figure 6, the debugger will recognize that first Boolean expression 22 ("(K > 999) && (K < 1050))" is invariant within the loop when the condition contained in line 8 ("if (FLAG == TRUE)") is not satisfied. Thus, the debugger may extract second Boolean expression
 15 24 as "(!(FLAG == TRUE))".

It should be noted that, as with first Boolean expression 22, second Boolean expression 24 may also be a compound Boolean expression. Further, first and second Boolean expressions 22, 24 may be extracted from a plurality of lines of code (e.g., from nested if-statements, combinations of if-statements and loop control expressions,
 20 etc.), and treated as a single Boolean expression for evaluation purposes.

In a special conditional breakpoint setting (blocks 4a-d), the debugger sets special conditional breakpoint 26 at a pre-ICB program position. A pre-ICB program position may be a program position that would be reached before, or when, the initial conditional breakpoint 20 is reached, but nonetheless, evaluated before initial
 25 conditional breakpoint 20 would be evaluated.

As shown in Figures 1 and 2, in a first class of embodiments, initial conditional breakpoint 20 is removed at the outset, and thereafter reestablished (blocks 10a and 10b) if special conditional breakpoint 26 is satisfied. Therefore, according to this first class of embodiments, special conditional breakpoint 26 may be defined to be satisfied when initial conditional breakpoint 20 would be satisfied (where first Boolean expression 20 is completely invariant within the loop); or when initial conditional breakpoint 20 would be satisfied or first Boolean expression 22 would be variant within the loop (where first Boolean expression is partially invariant within the loop).

Thus, according to the first class of embodiments, in Figure 5, the debugger sets, at line 4, special conditional breakpoint 26 (labeled as " SCB ") having first Boolean expression 22, as "BREAK 4 WHEN K > 999"; while in Figure 6, the debugger sets, at line 4, special conditional breakpoint (also labeled as " SCB "), having first Boolean expression disjunctively with the complement of second Boolean expression 24, as "BREAK 4 WHEN ((K > 999) && (K < 1050)) || (FLAG == TRUE))".

As shown in Figures 3 and 4, in a second class of embodiments, initial conditional breakpoint 20 is not removed at the outset, but removed (blocks 10c and 10d) if special conditional breakpoint 26 is satisfied. Therefore, according to the second class of embodiments, special conditional breakpoint 26 may be defined to be satisfied when initial conditional breakpoint 20 would not be satisfied (where first Boolean expression 20 is completely invariant within the loop); or when initial conditional breakpoint 20 would be not satisfied and first Boolean expression 22 would be invariant within the loop (where first Boolean expression is partially invariant within the loop).

Thus, according to the second class of embodiments, in Figure 5, the debugger sets, at line 4, special conditional breakpoint 26 (labeled as "SCB' ") having the complement of first Boolean expression 22, as "BREAK 4 WHEN !(K > 999)"; while in Figure 6, the debugger sets, at line 4, special conditional breakpoint (also labeled as "SCB' "), having the complement of first Boolean expression conjunctively with second Boolean expression 24, as "BREAK 4 WHEN (!(K > 999 && K < 1050)) && !(FLAG == TRUE))".

In an initial conditional breakpoint removal (blocks 6a-d and 12a-b), the debugger removes the initial conditional breakpoint from its program position within the loop.

It should be noted that a program loop may be encountered only once during execution of the program. However, a program loop may be encountered multiple times during execution of a program. Therefore, the present invention includes setting one or more optional reset breakpoints to accommodate program loops that may be encountered multiple times.

In an optional reset breakpoint setting (blocks 8a-d), the debugger may set a reset breakpoint 28 (discussed in more detail below) at one or more loop exit program positions. In Figure 5, the debugger sets a reset breakpoint 28 at line 8; while in Figure 6, the debugger sets a reset breakpoint at lines 11 and 14.

It should be noted that a loop exit program position may be a program position that would be reached after, or as, the loop finishes iterating, but nonetheless, before the special conditional breakpoint 26 is subsequently evaluated. A loop exit position may be within the loop itself, as is illustrated in Figure 6, in which a reset breakpoint 28 is set within the loop at line 11; or outside the loop, as is illustrated in Figure 5, in which a reset breakpoint is set outside the loop at line 8. Further, a loop exit position

may even share the position of special conditional breakpoint 26, as long as reset breakpoint 28 is processed before evaluation of special conditional breakpoint.

With the first exemplary class of embodiments, after removing initial conditional breakpoint 20 and setting special conditional breakpoint 26, the debugger
5 creates a state that assists in avoiding unnecessary evaluations of first Boolean expression 22 that may have occurred within the program loop via initial conditional breakpoint 20. Regarding the second exemplary class of embodiments, after setting special conditional breakpoint 26, the debugger creates a similar state. From either of these states, the debugger may allow the program to execute, during which special
10 conditional breakpoint 26 may be subsequently satisfied.

As shown in Figures 1 and 2, regarding the first class of embodiments, if special conditional breakpoint 26 is satisfied, through an initial conditional breakpoint reestablishment (blocks 10a and 10b) subsequent program execution may be halted as initially intended by the programmer at initial conditional breakpoint 20. Thus, the
15 programmer may debug the program as initially desired, and when ready, allow the program to further execute. When program execution reaches a reset breakpoint 28, an optional subsequent initial conditional breakpoint removal (blocks 12a and 12b) and/or an optional reset breakpoint removal (blocks 14a and 14b) may be effectuated leaving special conditional breakpoint 26 set for further breakpoint impact reduction.

Referring now to Figures 3 and 4, regarding the second exemplary class of embodiments, if special conditional breakpoint 26 is satisfied, through an initial conditional breakpoint removal (blocks 6c and 6d) the timing of subsequent program execution will not be affected by the unnecessary evaluation of initial conditional breakpoint 20. When program execution reaches a reset breakpoint 28, an optional
20 initial conditional breakpoint reestablishment (blocks 10c and 10d) and/or an optional
25

reset breakpoint removal (blocks 14c and 14d) may be effectuated leaving special conditional breakpoint 26 set for further breakpoint impact reduction.

It should be understood, however, that the invention is not necessarily limited to the specific process, order, arrangement and components shown and described

5 above, and may be susceptible to numerous variations within the scope of the invention. For example, although the above-described exemplary aspects of the invention are believed to be particularly well suited for reducing debugger impact with for-loops and while-loops, it is contemplated that the concepts of the present invention can be adapted for various types of programming loop and programming
10 language syntaxes. For example, the concepts of the present application can be utilized with a do-while-loop, a goto-loop, etc., in Assembly, Basic, C, C++, etc.

It will be apparent to one skilled in the art that the manner of making and using the claimed invention has been adequately disclosed in the above-written description of the embodiments taken together with the drawings. Further, it will be
15 apparent that the above methods may be readily embodied in a method, a software debugger, and a computer program medium, which may be any form of a computer program medium readable by a computer, such as, for example, a CD-ROM, RAM, a hard drive, a cache, etc.

It will be understood that the above description of the embodiments of the
20 present invention are susceptible to various modifications, changes, and adaptations, and the same are intended to be comprehended within the meaning and range of equivalents of the appended claims.